Combining optimal control and learning for autonomous aerial navigation in novel indoor environments

Kevin Lin, Brian Huo, Megan Hu

***Abstract*— This report proposes a combined optimal control and perception framework for Micro Aerial Vehicle (MAV) autonomous navigation in indoor enclosed environments. We demonstrate the efficacy of the framework across novel scenes in the iGibson simulation environment.**

I. INTRODUCTION

Aerial navigation is a well-studied problem in the literature. Researchers have taken advantage of deep learning's ability to implicitly understand visual cues to enable quadrotors to autonomously navigate a variety of environments. However, current solutions to this problem fail to take advantage of aerial robots' ability to move in 3D space. Robots should be able to fly over, under, and around obstacles in 3D space rather than restricting themselves to flying at a prespecified height.

In this paper, we present an algorithm for goal-driven 3D navigation in novel cluttered indoor environments using first-person view images, under the assumption of perfect state estimation. By marrying deep learning and optimal control, we are able achieve good performance in a sample-efficient manner, and translate from simulation to hardware with no additional fine tuning.

II. RELATED WORK

We base our work off Bansal *et al.*'s LB-WayPtNav [1], which demonstrated better performance than end-to-end navigation methods by learning the input to a model-based planning and control module rather than directly learning a control policy. Xia *et al.* used a similar pipeline for control on serial chain manipulators [13]. We extend this methodology to 3D navigation.

There have been a number of other works that use deep learning for navigation of aerial vehicles through roads [6], trails [12], and indoor environments [3], using first person view images. However, none of these other works perform point-based navigation, or allow the drone to change its height. Kauffman *et al.*'s approach is the closest to ours; they learn waypoints to navigate between gates in a 2D drone race track [4]. However, they do not consider obstacles or allow for 3D motion.

The contributions of this paper are as follows. First, we demonstrate how to leverage existing approaches that combine optimal control and learning based methods for true $3D$ goal-driven quadrotor flight. Second, we demonstrate that the learned policy generalizes to novel environments in simulation.

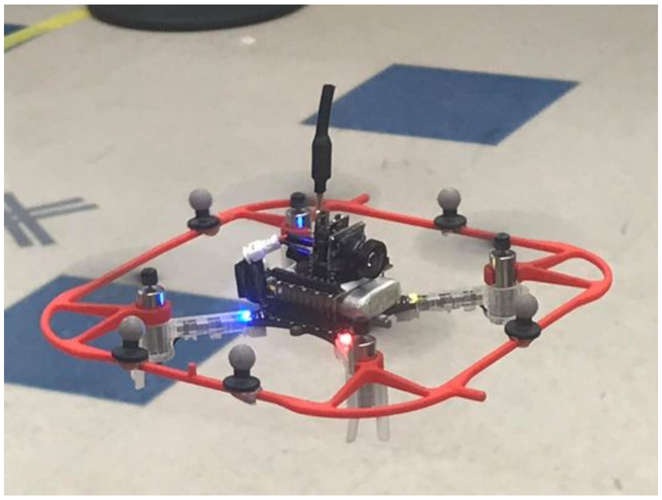


Fig. 1: A sample crazyflie 2.0 test platform, with onboard RGB camera and optitrack beacons.

III. PROBLEM FORMULATION

We study the problem of autonomous aerial robot navigation in *a priori* unknown indoor environments. We specifically focus on situations where true 3D planning is required. For example, an aerial robot may have to fly over a table, under an overhang, and around a coat rack to reach its destination. The robot's task is to fly to some fixed point in space $p^* = [x^*, y^*, z^*]^T$ without colliding with the environment.

We take our robot to be a quadrotor with a single, forward-facing, monocular RGB camera mounted at a fixed height and pitch from the center of mass. The quadrotor's state can be described by $p = [x, y, z]^T$, the position of the robot's center of mass in the world coordinate frame, R a rotation matrix describing the robot's orientation with respect to this same frame, and their derivatives $\dot{p} = [\dot{x}, \dot{y}, \dot{z}]^T$ and $\dot{R}$. The dynamics of a quadrotor can be found in section II of [5]. We do not consider state estimation in this work, and assume that the robot state, and thus the robot's relative pose to the goal, are perfectly known.

For planning purposes, we use a spline-based trajectory planner, though any other trajectory planner such as minimum snap [7] may be used. Our trajectory planner uses a differentially flat representation of the quadrotor model with outputs $p = [x, y, z]^T$ and $\psi = \arctan 2(R_{3,2}, R_{2,3})$, the yaw angle of the robot. Given the robot's current state and a waypoint in these coordinates, $w = [w_x, w_y, w_z, w_\psi]^T$, the planner computes the trajectory to the waypoint which minimizes the snap, or fourth derivative, in x, y, and z, as well as the angular acceleration $\ddot{\psi}$ along the trajectory.

In this work, we learn a navigation policy which outputs a waypoint for this min-snap trajectory planner in order to bring the robot to the goal p^* while avoiding obstacles. At a given time t, the policy is given an input $(I, x_{rel}, y_{rel}, z_{rel}, \dot{x}_{rel}, \dot{y}_{rel}, \dot{z}_{rel}, \dot{\psi})$ and outpoints a waypoint $w = [w_x, w_y, w_z, w_\psi]^T$. Here, I is the current RGB image from the robot's camera, and $[x_{rel}, y_{rel}, z_{rel}] = R^T(p^* - p)$ and $[\dot{x}_{rel}, \dot{y}_{rel}, \dot{z}_{rel}]^T = R^T \dot{p}$ are the relative position and velocity to the goal, represented *in the body frame* of the robot (not the world coordinate frame). The tasks are performed in novel indoor environments that the robot has never encountered, and whose map or topology are not given to the robot.

IV. Model based learning for 3D point goal visual navigation

Like the visual navigation technique developed by Bansal *et al.* [1], our approach uses three modules for 3D point goal visual navigation: perception, planning and control.

A. Learning based waypoint generation for 3D point goal visual navigation

1) Perception: We use a Convolutional Neural Network (CNN) pretrained for a surface normals vision task [9] to learn the quadrotor's next waypoint. Specifically, the CNN takes in a 256×256 pixel RGB image from the onboard camera, the position of the goal in the robot's local coordinate frame and yaw velocity of the robot, and outputs a waypoint w. The CNN is trained purely in simulation using an expert policy which uses full knowledge of the world to generate an 'optimal' waypoint.

2) Planning: Before passing the waypoint into the planner, we first preprocess it by clipping it into the camera's vision cone. If the waypoint is far outside the cone, such as behind the drone, this generally results in the drone spinning in place. This preprocessing implicitly forces the policy to "look before it leaps" and prevents the drone from colliding with unobserved obstacles.

We represent a trajectory as a vector of time-parameterized continuous functions in the flat coordinates: $\xi(t) = [x(t), y(t), z(t), \psi(t)]$, where $t = 0$ represents the current state, and $t = T$ represents the state at trajectory completion, ie the desired waypoint. Given the current drone state $[p_0, \psi, \dot{p}_0, \dot{\psi}_0]$ and a waypoint $[w_x, w_y, w_z, w_\psi]$, the spline-based trajectory planner solves a set of linear equations to output a trajectory in these flat coordinates.

3) Control: One benefit of our pipeline is that it's controller-agnostic. As long as the controller is able to reliably track the trajectory planner's, we can use any controller. This also means that our learned policy generalizes well to other (similarly-sized) drones. In our simulated experiments we use a PID controller from Panerati *et al.* [8], while in our hardware experiments, we use an LQR controller from [2]. The controller is executed for a time horizon of H seconds (which likely does not lead to completing the trajectory), after which a new image is taken and the process begins again.

B. Data generation and training procedure

We train the perception module entirely in simulation using a variant of Dataset Aggregation, where the expert action is automatically generated using a combination of model predictive control, A^* planning and full knowledge of the simulation environment. The simulation environment comes from the iGibson [11] dataset (where the simulation engine is PyBullet [8]), which contains a large variety of 3D scanned environments. To generate training data, we first randomly sample start and goal points for the robot. Then, we let the robot take the expert waypoint with probability $\alpha \in [0, 1]$ and the current CNN output waypoint with probability $1 - \alpha$, where α decays to 0. We continue until either the robot reaches the goal if until the robot collides. After we collect X new data points, we train the current network on the new dataset $D = D \cup X$. A new single data point consists of the image, goal position, velocities and also the *expert* action. Then, we repeat this process of aggregating more data to the dataset and training the policy on the larger dataset. For each of the scenes in simulation, we collect 20k data points in this manner. **CNN architecture Details** We use the standard encoder-decoder architecture of a ResNet-50 [14] backbone. For the encoder, we use that of a ResNet-50 pretrained on the surface normals task. We freeze this encoder and only train the decoder part of the network which consists of 5 fully connected layers with ReLU activations. We use an MSE loss and AdamW optimizer with learning rate 10^{-4} and weight decay coefficient of 10^{-2} on 80k data samples collected using the DAgger procedure described above.

C. Expert Policy

To generate 'optimal' waypoints for training, we score their resultant trajectories using an MPC-esque cost function which balances distance from obstacles with distance to the goal. Given a prospective waypoint w, we compute the corresponding trajectory $\xi(t) = [p(t), \psi(t)]$, and evaluate it using the corresponding cost function:

$$J(w) = \lambda_1 J_{obs}(\xi) + \lambda_2 J_{dist}(\xi) + \lambda_3 J_{angle}(\xi) \quad (1)$$

$$J_{obs}(\xi) = -\max(d^{obs}_{cutoff} - d^{obs}(\xi)) \quad (2)$$

$$J_{dist}(\xi) = d^{goal}(\xi(0))^2 - d^{goal}(\xi(T))^2 \quad (3)$$

$$J_{angle}(\xi) = \frac{1}{T} \sum_{t=0}^{T} d^{angle}(\xi(t)) \quad (4)$$

The three cost terms respectively represent a penalty for closeness to obstacles, a penalty on the negative progress made to the goal and a penalty on the difference in angle between the 'optimal' angle. d^{obs} represents the signed distance to the nearest static obstacle. $d^{goal}(\xi(T))$ represents the minimum collision-free distance between $\xi(T)$ and the goal location. $d^{angle}(\xi(t))$ represents the difference between the angle produced by the trajectory planner at time t and the 'optimal' angle for the trajectory position. We compute the distances to the goal using the Fast Marching Method (FMM) [10] and compute 'optimal' angles for points along

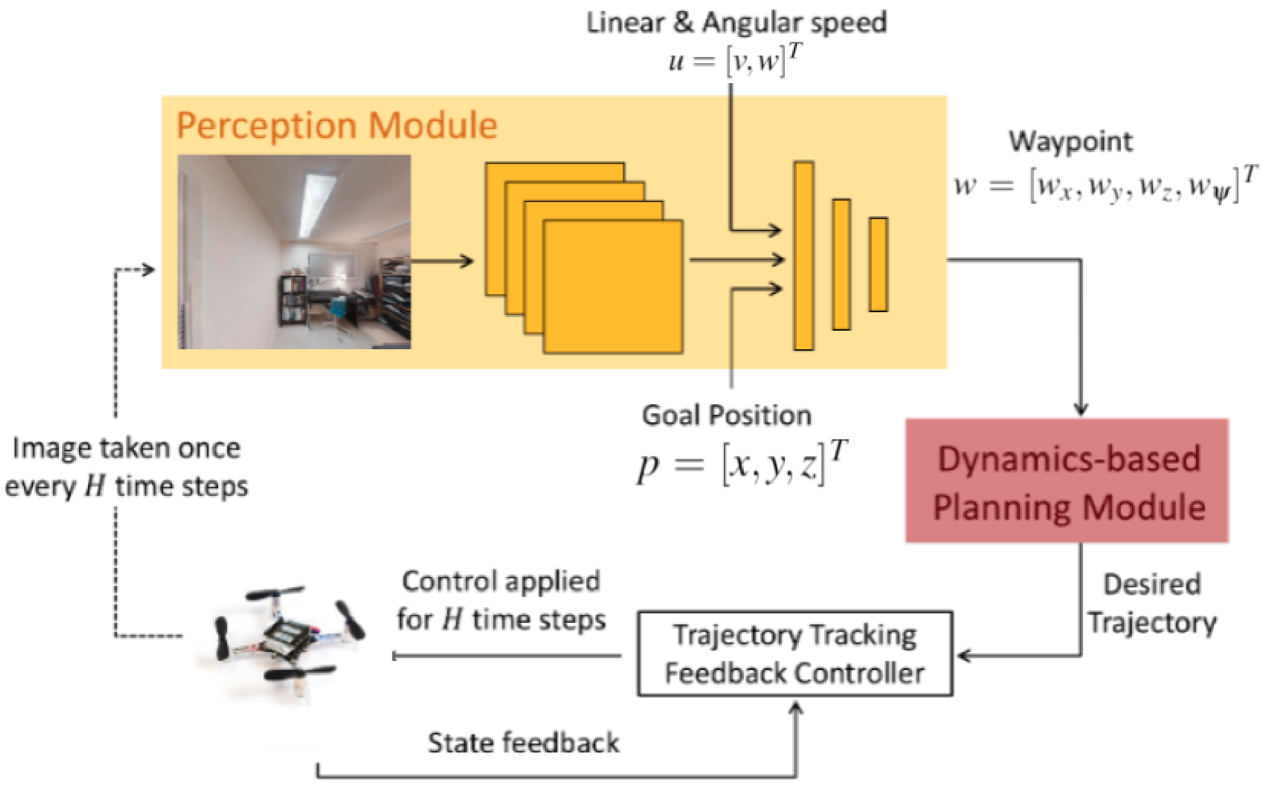

Linear & Angular speed
$u = [v, w]^T$
Perception Module
Waypoint
$w = [w_x, w_y, w_z, w_\psi]^T$
Goal Position
$p = [x, y, z]^T$
Image taken once every H time steps
Dynamics-based Planning Module
Control applied for H time steps
Desired Trajectory
Trajectory Tracking Feedback Controller
State feedback

Fig. 2: The drone control pipeline relies on a cycle between the perception module, planning module, and flight controller. The perception module takes in RGB images, linear and angular velocity measurements, and a goal position relative to the drone, then uses a neural network to output a waypoint in xyz-yaw coordinates. This waypoint is passed to the planning module, which plots a trajectory to the waypoint. A PID feedback controller tracks the execution of the trajectory as informed by state feedback. Images for the perception module are continuously taken while in flight.

a trajectory by taking gradients on the distance potentials produced by the FMM map. The λ_i terms denote weights for each term. Here, we use $d^{obs}_{cutoff} = 0.4m$, $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 0.2$.

We select a waypoint by sampling waypoints and choosing the one with the lowest cost using the MPC cost function outlined above. Since the sample space is large, we increase sample efficiency by biasing our sampling around the output of an A^* planner. However, this planner empirically serves as a good heuristic for sampling.

V. Experimental Design

A. Simulation Experiments

For simulation experiments, we used photo-realistic indoors scenes from the iGibson dataset [11] and use the CrazyFlie 2.1 model from *gym-pybullet-drones*. We trained the CNN using 4 indoor scenes and test on a 5th held out scene. For the testing scene, we run our policy on 200 randomly sampled test episodes (start, goal position pairs) where the start and goal position pairs involved scenarios like turning a corner, flying above some obstacle(s) and leaving a room. We call a trial successful if the robot reaches reaches within 0.3m inside the goal point without collisions. For the simulation experiments, we find that the network is able to successfully generalize to the novel unseen scenes.

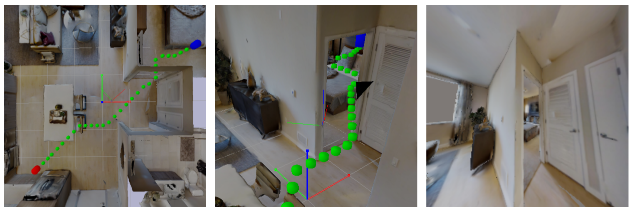
TABLE I: Results on simulation experiments

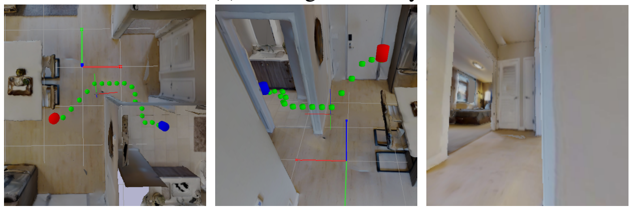
Agent	**Expert**	**LBWayPt3D (Training)**	**LBWayPt3D (Novel)**
Success Rate (%)	100	93%	76%

Failure Modes. Similar to Bansal *et al.* [1], though our policy is able to perform navigation tasks in novel environments, it can only do local reasoning as there is no memory or map implemented in the network. The most prominent failure modes are: a) when the robot is too close to an obstacle, and b) situations that require 'backtracking' from an earlier planned path.

VI. Conclusions and Future Work

In this report, we present a framework for autonomous aerial goal navigation by combining optimal control techniques and learning methods by building off the LBWayPt-Nav framework [1]. Similar to [1], we demonstrate, the efficacy of combining optimal control methods with learning methods in simulation. On the other hand, the framework also assumes perfect state estimation and employs a purely reactive policy. These assumptions are not optimal for long range tasks, where incorporating long-term spatial memory is often critical.


(a) Through doorway


(b) Around corner

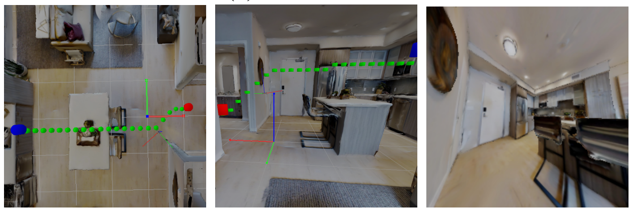

(c) Over table

Fig. 3: Representative simulation experiment episodes on the held out testing scene. Images for each row are given in the order of: bird's eye view, third person view, and robot first person view. The red dot represents the starting point, the blue dot represent the end goal, the green dots represents the path of an A^* planner.

VII. Acknowledgements

We like to thank UC Berkeley's EECS 106B for being the reason we started working on this project, Somil Bansal and the authors of LBWayPtNav for inspiring this particular project, Valmik Prahbu, Somil Bansal and Shankar Sastry for discussions and advice, and David Fridovich-Keil, David McPherson, and Joseph Menke for help with the Crazyflies and Optitrack.

Appendix

References

[1] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," 2019.

[2] D. Fridovich-Keil, J. Fisac, and A. Bajcsy, "crazyflie_clean source code," https://github.com/HJReachability/crazyflie_clean, 2019, accessed: 2021-09-14.

[3] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," *CoRR*, vol. abs/1902.03701, 2019. [Online]. Available: http://arxiv.org/abs/1902.03701

[4] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Conference on Robot Learning*. PMLR, 2018, pp. 133–145.

[5] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

[6] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, 2018.

[7] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[8] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," 2021.

[9] A. Sax, B. Emi, A. R. Zamir, L. J. Guibas, S. Savarese, and J. Malik, "Mid-level visual representations improve generalization and sample efficiency for learning active tasks," *CoRR*, vol. abs/1812.11971, 2018. [Online]. Available: http://arxiv.org/abs/1812.11971

[10] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996. [Online]. Available: https://www.pnas.org/content/93/4/1591

[11] B. Shen*, F. Xia*, C. Li*, R. Martín-Martín*, L. Fan, G. Wang, S. Buch, C. D'Arpino, S. Srivastava, L. P. Tchapmi, K. Vainio, L. Fei-Fei, and S. Savarese, "igibson, a simulation environment for interactive tasks in large realistic scenes," *arXiv preprint arXiv:2012.02924*, 2020.

[12] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," *CoRR*, vol. abs/1705.02550, 2017. [Online]. Available: http://arxiv.org/abs/1705.02550

[13] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation," *arXiv preprint arXiv:2008.07792*, 2020.

[14] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: Multilevel residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, p. 1303–1314, Jun 2018. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2017.2654543